**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Параллельные алгоритмы»**

Тема: IPC в UNIX

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Митяков А.В. |

Санкт-Петербург

2018

**Цель работы.**

Знакомство с IPC в UNIX.

**Задание.**

Параллельное вычисление числа Пи



Входные данные для программы:

* n - Количество слагаемых
* k - Количество процессов

**Основные теоретические положения.**

Классические средства межпроцессорного взаимодействия:

* Переменные окружения
* Сигналы
* Каналы
  + Именованные
  + Неименованные
* Сокеты

Переменная среды (англ. environment variable) — текстовая переменная операционной системы, хранящая какую-либо информацию — например, данные о настройках системы.

Сигналы - программные прерывания обеспечивающие асинхронную обработку событий. Возможные действия на событие:

* Игнорировать сигнал (кроме SIGKILL и SIGSTOP)
* Перехватить сигнал и обработать его
* Действие по умолчанию

Канал — поток данных между двумя или несколькими процессами, имеющий интерфейс, аналогичный чтению или записи в файл. Каналы имеют два ограничения:

1. Исторически они являются симплексными (то есть данные могут передаваться по ним только в одном направлении).
2. Каналы могут использоваться только для организации взаимодействия между процессами, которые имеют общего предка. Обычно канал создается родительским процессом, который затем вызывает функцию fork, после чего этот канал может использоваться для общения между родительским и дочерним процессами.

**Ход работы.**

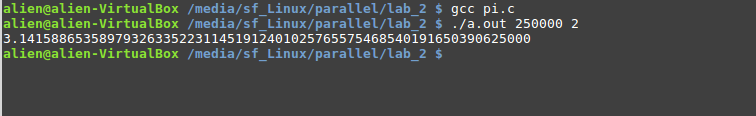
1. Воспользуемся ассоциативностью операции сложения. Каждый процесс может считать отдельную часть суммы, которая приближает число пи. Далее главный процесс агрегирует полученные дочерними процессами результаты.
2. Для взаимодействия процессов будем использовать каналы, т.к. обмениваться информацией будут только родительский и дочерние процессы, а также родительский процесс будет только читать, а дочерний только писать.
3. Программа будет работать следующим образом:
   1. Определение количества слагаемых на процесс.
   2. В цикле длинной в количество используемых процессов системный вызов *pipe(int pipefd[2])* для создания канала и *fork()* для создания дочернего процесса. Далее в дочернем процессе суммирование отведенного данному процессу слагаемых, а в родительском следующая итерация цикла и создание следующего процесса.
   3. В родительском процессе после завершения цикла создания процессов системные вызовы *waitpid(-1, NULL, 0)* для ожидания завершения всех дочерних процессов.
   4. Чтение из всех созданных каналов и суммирования результатов.
4. Скомпилируем программу **pi.c**. Программа принимает на вход два числа – первое число слагаемых в ряду, а второе – количество процессов для работы. В результате на выводится полученное значение с точностью 64 знака после запятой.

Рисунок 1 – Компиляция и запуск программы расчёта

В данном примере программы запускается с параметрами **250000** и ***2***, что означает использовать 250 тысяч слагаемых и 2 процесса.

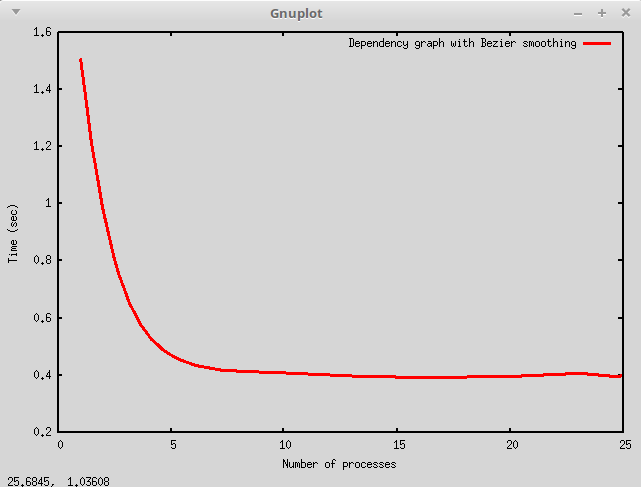
1. Далее для измерения времени работы программы для данного количества процессов, а также для перебора различного количества процессов и анализа зависимости времени исполнения от количества используемых процессов составим программу **run.sh** и запустим её. В результате получим следующий график:

Рисунок 2 – Зависимость времени исполнения от количества процессов со сглаживанием

Как видно из графика с ростом количества процессов время исполнения уменьшается, однако при использовании более 4 процессов время исполнения программы практически не меняется, что можно объяснить наличием 4 логических процессоров в машине, на которой запускалась программа. Исходный код приведён в приложениях А, Б. В приложении В приведена таблица зависимости времени выполнения программы от количества используемых процессоров.

**Выводы**

Классические средства межпроцессорного взаимодействия обеспечиваются возможность взаимодействия процессов. Переменные окружения позволяют передать информацию, процессу. Каналы обеспечивают взаимодействие только родительского и дочернего процесса и только в одном направлении. Таким образом есть существенные ограничения.

Приложение А

PI.C

#include "stdio.h"

#include "unistd.h"

#include "errno.h"

#include "sys/types.h"

#include "sys/wait.h"

#include "stdlib.h"

long double calcPartSeries(int start, int end){

long double result = 0.0;

for (int i = start; i <= end; i++)

result += (long double)(((i + 1) % 2 ? -1 : 1) \* 4) / (long double)(2 \* i - 1);

return result;

}

int main(int argc, char const \*argv[])

{

int numOfItems = atoi(argv[1]);

int numOfProcesses = atoi(argv[2]);

int itemsPerProcess = numOfItems / numOfProcesses;

int\*\* channels = (int\*\*)malloc(sizeof(int\*) \* numOfProcesses);

for (int i = 0; i < numOfProcesses; i++) {

channels[i] = (int\*)malloc(sizeof(int) \* 2);

}

for (int i = 0; i < numOfProcesses; i++) {

int res = pipe(channels[i]);

if (res < 0){

printf("Error of pipe\n");

return 1;

}

pid\_t pid = fork();

if (pid == 0) {

close(channels[i][0]);

long double current\_res = calcPartSeries(i \* itemsPerProcess + 1, (i + 1) \* itemsPerProcess);

write(channels[i][1],&current\_res,sizeof(long double));

return 0;

}

else if (pid > 0) {

close(channels[i][1]);

continue;

}

else {

printf("Error of fork\n");

return 1;

}

}

pid\_t pid;

while (pid = waitpid(-1, NULL, 0)) {

if (errno == ECHILD) {

break;

}

}

long double result = 0.0;

long double part = 0.0;

for (int i = 0; i < numOfProcesses; i++) {

read(channels[i][0], &part, sizeof(long double));

result += part;

}

printf("%.64Lf\n", result);

for (int i = 0; i < numOfProcesses; i++) {

free(channels[i]);

}

free(channels);

return 0;

}

Приложение Б

run.sh

#!/bin/bash

gcc pi.c

echo -e "# Num proc\tTime" > plot.txt

for (( i=1; i<=25; i++))

do

echo -n "Calculate using ${i} proc: "

(time ./a.out 4655851200 ${i}) &> file.txt

echo -ne "${i}\t" >> plot.txt

time=`head -3 file.txt | grep -o -P '(?<=m).\*(?=s)'`

echo "${time}"

echo "${time}" >> plot.txt

done

rm -f file.txt a.out

gnuplot -e "set ylabel \"Time (sec)\"; set xlabel \

\"Number of processes\"; plot \"plot.txt\" with lines \

smooth sbezier lw 3 t \"Dependency graph with Bezier smoothing\"; \

pause -1"

Приложение В

**зависимость времени выполнения от количества процессоров**

# Num proc Time

1 1.469

2 0.774

3 0.511

4 0.382

5 0.490

6 0.461

7 0.403

8 0.385

9 0.398

10 0.379

11 0.384

12 0.374

13 0.379

14 0.382

15 0.384

16 0.389

17 0.386

18 0.394

19 0.375

20 0.381

21 0.400

22 0.386

23 0.398

24 0.385

25 0.386